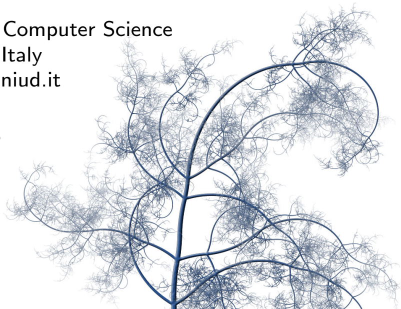
A Contraction Method to Decide MSO Theories of Trees

Gabriele Puppis

Departement of Mathematics and Computer Science
University of Udine, Italy
gabriele.puppis@dimi.uniud.it

Verona 2007



What is the talk about?

An **automaton-based** approach to solve **model-checking problems** for **monadic second-order logics** over (a large class of) **trees**.

What is the talk about?

An **automaton-based** approach to solve **model-checking problems** for **monadic second-order logics** over (a large class of) **trees**.

We shall briefly explain what we mean by

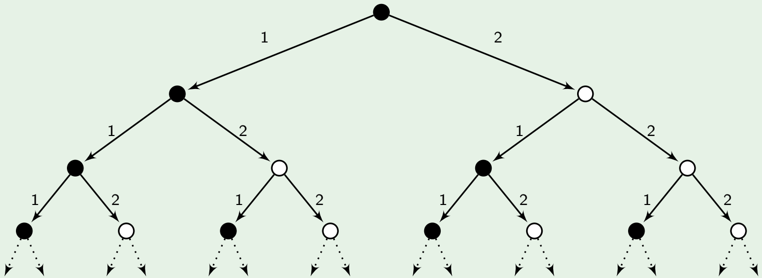
- **tree**
- **monadic second-order (MSO) logic**
- **model-checking problem**
- **automaton-based approach**.

We shall consider possibly infinite (rooted unranked) trees where

- each vertex is associated a **color** (e.g., black, white)
- each edge is associated a **label** (e.g., 1, 2)
- edges departing from the same vertex have different labels (deterministic trees).

Example

The infinite $\{1, 2\}$-labeled $\{black, white\}$-colored tree:



Definition (MSO logic)

Given a tree $T = (V, (E_a)_{a \in A}, (P_c)_{c \in C})$,
MSO-formulas over T are defined as follows:

- node variables $x, y, z, \ldots$ denote single elements in V
- set variables $X, Y, Z, \ldots$ denote subsets of V
- atomic formulas have one of the following forms:
  - $E_a(x, y)$ meaning '(x, y) *denotes an a-labeled edge*'
  - $P_c(x)$ meaning 'x *denotes a c-colored vertex*'
  - $X(y)$ meaning 'y *denotes a vertex in the set* X'
- more complex formulas are build up via
  - the Boolean connectives $\wedge$, $\vee$, $\neg$
  - quantifications $\exists x, \forall x$ over node variables
  - quantifications $\exists X, \forall X$ over set variables

Example 1

The *reflexive and transitive closure* E^* of a successor relation E is definable in MSO logic:

$$E^*(x, y) \ := \ \forall X.\, X(x) \ \wedge \ \forall z, w.\, \big(X(z) \wedge E(z, w) \rightarrow X(w)\big) \ \rightarrow \ X(y)$$

Example 1

The *reflexive and transitive closure* E^* of a successor relation E is definable in MSO logic:

$$E^*(x,y) \;:=\; \forall\, X.\, X(x) \;\wedge\; \forall\, z,w.\, \big(X(z) \wedge E(z,w) \rightarrow X(w)\big) \;\rightarrow\; X(y)$$

Example 2

'*Any two vertices have a common ancestor*'
is translated into

$$\forall\, x,y.\, \exists\, z.\, E^*(z,x) \;\wedge\; E^*(z,y)$$

Example 1

The *reflexive and transitive closure* E^* of a successor relation E is definable in MSO logic:

$$E^*(x,y) \ := \ \forall\, X.\, X(x) \ \wedge \ \forall\, z,w.\, \big(X(z) \wedge E(z,w) \rightarrow X(w)\big) \ \rightarrow \ X(y)$$

Example 2

'*Any two vertices have a common ancestor*'
is translated into

$$\forall\, x,y.\, \exists\, z.\, E^*(z,x) \ \wedge \ E^*(z,y)$$

Example 3

'*One can always reach a bad vertex from a good one*'
is translated into

$$\forall\, x.\, P_{good}(x) \ \rightarrow \ \exists\, y.\, P_{bad}(y) \ \wedge \ E^*(x,y)$$

Note that we can get rid of node variables $x, y, z, \ldots$ by simulating them via set (singleton) variables $X, Y, Z, \ldots$.

Given a tree T with vertices colored over $\{c_1, \ldots, c_n\}$, we are interested in solving the following problem, denoted $\mathbf{MTh}_T$:

Definition (model-checking problem)

Input: a formula φ with free set variables $X_1, \ldots, X_n$

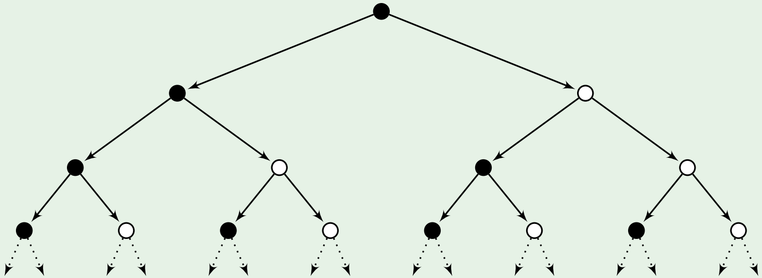
Problem: to decide whether φ holds in T (shortly, $T \models \varphi$) by interpreting each variable X_i with the set of c_i-colored vertices.

Example

Check whether the formula

$$\varphi(X) = X(root) \wedge \forall x, y.\ \big(E_{left}(x, y) \rightarrow X(y)\big)$$

holds in the following tree by interpreting
X with the set of *black-colored vertices*:



We solve the model-checking problem by means of automata ...

Definition (Rabin tree automaton)

A Rabin tree automaton running on *A-labeled C-colored trees* is a tuple $\mathcal{A} = \big(Q, \Delta, \mathcal{I}, \{p_1, ..., p_k\}\big)$, where

- Q is a finite set of states
- $\Delta \subseteq Q \times C \times Q^A$ is a transition relation
- $\mathcal{I} \subseteq Q$ is a set of initial states
- each p_i is an **accepting pair** $\big(Good_i, Bad_i\big)$, with $Good_i, Bad_i \subseteq Q$.

We solve the model-checking problem by means of automata ...

Definition (Rabin tree automaton)

A Rabin tree automaton running on *A-labeled C-colored trees* is a tuple $\mathcal{A} = \big(Q, \Delta, \mathcal{I}, \{p_1, ..., p_k\}\big)$, where

- Q is a finite set of states
- $\Delta \subseteq Q \times C \times Q^A$ is a transition relation
- $\mathcal{I} \subseteq Q$ is a set of initial states
- each p_i is an **accepting pair** $\big(Good_i, Bad_i\big)$, with $Good_i, Bad_i \subseteq Q$.

... But, how does a Rabin tree automaton *run* on a tree?

First, the automaton $\mathcal{A}$ non-deterministically generates a **computation** on the input tree T:

- it marks the root of T with any arbitrary state
- it marks the successors of each vertex of T on the basis of the current color and the transition relation Δ.

First, the automaton $\mathcal{A}$ non-deterministically generates a **computation** on the input tree T:

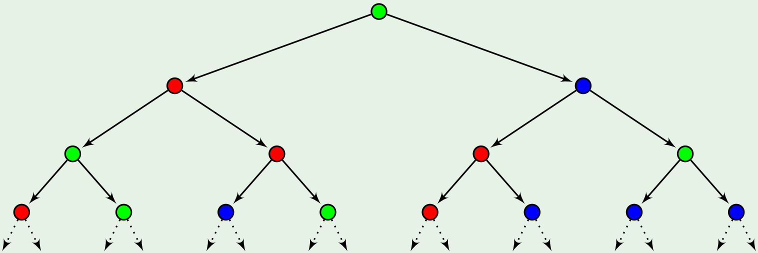
- it marks the root of T with any arbitrary state
- it marks the successors of each vertex of T on the basis of the current color and the transition relation Δ.

Then, it checks whether the computation is **successful**:

- the state at the root should be an *initial state*
- for every infinite path π,
  there should be a pair $p_i = \left(Good_i, Bad_i\right)$ such that
  (i) *at least one state* in $Good_i$ occurs *infinitely often* in π
  (ii) *every state* in Bad_i occurs *only finitely often* in π.

Example

Consider a $\{blue, red, green\}$-colored tree



and a Rabin tree automaton having

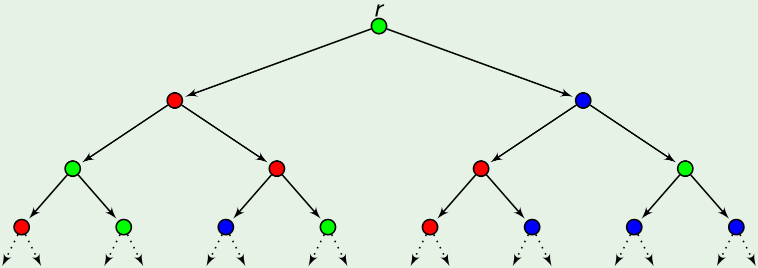
- three states b, r, g, that keep track of which color was seen last
- transitions

$$\begin{array}{lll} (b, blue, b, b) & (r, blue, b, b) & (g, blue, b, b) \\ (b, red, r, r) & (r, red, r, r) & (g, red, r, r) \\ (b, green, g, g) & (r, green, g, g) & (g, green, g, g) \end{array}$$

- a single accepting pair $p_1 = \big(Good_1, Red_1\big)$, with $Good_1 = \{b\}$ and $Red_1 = \{r\}$

Example

Consider a $\{blue, red, green\}$-colored tree

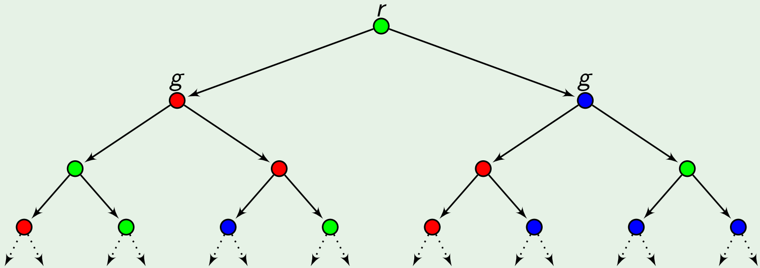


and a Rabin tree automaton having

- three states b, r, g, that keep track of which color was seen last
- transitions $\begin{array}{lll} (b, blue, b, b) & (r, blue, b, b) & (g, blue, b, b) \\ (b, red, r, r) & (r, red, r, r) & (g, red, r, r) \\ (b, green, g, g) & (r, green, g, g) & (g, green, g, g) \end{array}$
- a single accepting pair $p_1 = (Good_1, Red_1)$, with $Good_1 = \{b\}$ and $Red_1 = \{r\}$

Example

Consider a $\{blue, red, green\}$-colored tree

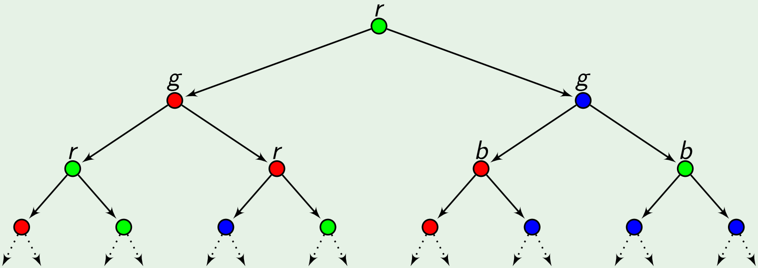


and a Rabin tree automaton having

- three states b, r, g, that keep track of which color was seen last
- transitions
$$\begin{array}{lll} (b, blue, b, b) & (r, blue, b, b) & (g, blue, b, b) \\ (b, red, r, r) & (r, red, r, r) & (g, red, r, r) \\ (b, green, g, g) & (r, green, g, g) & (g, green, g, g) \end{array}$$
- a single accepting pair $p_1 = (Good_1, Red_1)$, with $Good_1 = \{b\}$ and $Red_1 = \{r\}$

Example

Consider a $\{blue, red, green\}$-colored tree

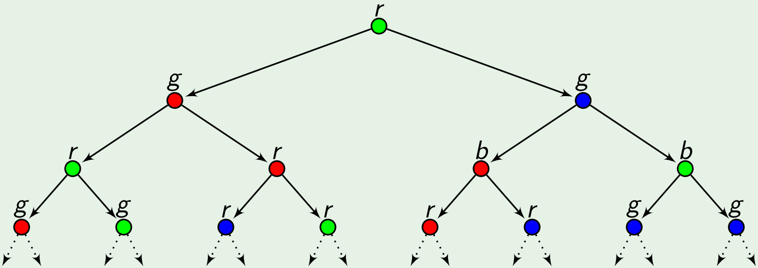


and a Rabin tree automaton having

- three states b, r, g, that keep track of which color was seen last
- transitions $\begin{array}{lll} (b, blue, b, b) & (r, blue, b, b) & (g, blue, b, b) \\ (b, red, r, r) & (r, red, r, r) & (g, red, r, r) \\ (b, green, g, g) & (r, green, g, g) & (g, green, g, g) \end{array}$
- a single accepting pair $p_1 = \big(Good_1, Red_1\big)$, with $Good_1 = \{b\}$ and $Red_1 = \{r\}$

Example

Consider a {*blue*, *red*, *green*}-colored tree



and a Rabin tree automaton having

- three states b, r, g, that keep track of which color was seen last
- transitions $(b, \textit{blue}, b, b)$ $(r, \textit{blue}, b, b)$ $(g, \textit{blue}, b, b)$
  $(b, \textit{red}, r, r)$ $(r, \textit{red}, r, r)$ $(g, \textit{red}, r, r)$
  $(b, \textit{green}, g, g)$ $(r, \textit{green}, g, g)$ $(g, \textit{green}, g, g)$
- a single accepting pair $p_1 = \big(\textit{Good}_1, \textit{Red}_1\big)$,
  with $\textit{Good}_1 = \{b\}$ and $\textit{Red}_1 = \{r\}$

Theorem (Rabin 1969)

Given any MSO-formula φ with free variables $X_1, \ldots, X_n$, one can compute a Rabin tree automaton $\mathcal{A}$ such that for every tree T with vertices colored over $\{c_1, \ldots, c_n\}$

$$\varphi \text{ holds in } T \qquad \text{iff} \qquad \mathcal{A} \text{ accepts } T$$

Theorem (Rabin 1969)

Given any MSO-formula φ with free variables $X_1, \ldots, X_n$, one can compute a Rabin tree automaton $\mathcal{A}$ such that for every tree T with vertices colored over $\{c_1, \ldots, c_n\}$

$$\varphi \text{ holds in } T \qquad \text{iff} \qquad \mathcal{A} \text{ accepts } T$$

$\Rightarrow$ Given a tree T, the following problem, denoted $\mathbf{Acc}_T$, becomes crucial:

Definition (acceptance problem)

Input: a Rabin tree automaton $\mathcal{A}$

Problem: to decide whether $\mathcal{A}$ accepts T (shortly, $T \in \mathscr{L}(\mathcal{A})$).

Proposition

The acceptance problem of any **regular** *tree T is decidable.*

Proposition

The acceptance problem of any **regular** *tree* T *is decidable.*

Proof sketch

- a regular tree T is bisimilar to a *finite graph*
- use this graph to produce a Rabin tree automaton $\mathcal{B}$ such that $\mathscr{L}(\mathcal{B}) = \{T\}$, namely, $\mathcal{B}$ accepts only T
- given any Rabin tree automaton $\mathcal{A}$,
  test whether $\mathscr{L}(\mathcal{A}) \cap \mathscr{L}(\mathcal{B})$ is non-empty.

Proposition

The acceptance problem of any **regular** *tree T is decidable.*

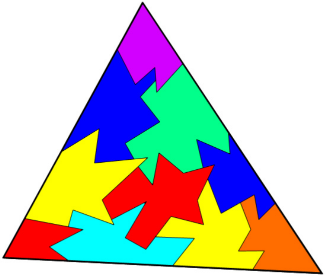
Proof sketch

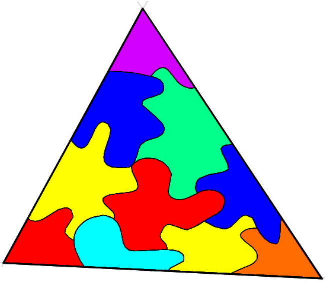
- a regular tree T is bisimilar to a *finite graph*
- use this graph to produce a Rabin tree automaton $\mathcal{B}$ such that $\mathscr{L}(\mathcal{B}) = \{T\}$, namely, $\mathcal{B}$ accepts only T
- given any Rabin tree automaton $\mathcal{A}$, test whether $\mathscr{L}(\mathcal{A}) \cap \mathscr{L}(\mathcal{B})$ is non-empty.

Problem

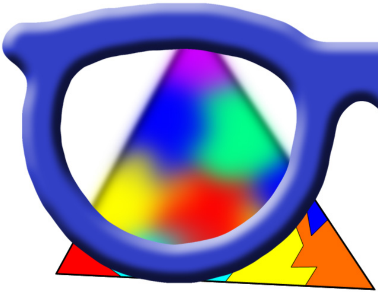
What about **non-regular** trees?

An automaton $\mathcal{A}$ can only **distinguish** between finitely many trees!

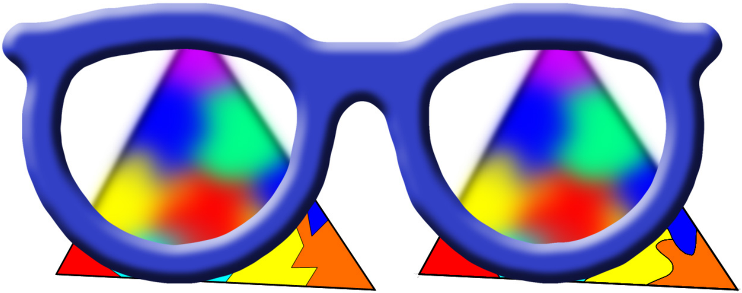




An automaton $\mathcal{A}$ can only **distinguish** between finitely many trees!



An automaton $\mathcal{A}$ can only **distinguish** between finitely many trees!



$\Rightarrow$ This allows us to introduce an **equivalence relation** $\equiv_{\mathcal{A}}$ such that

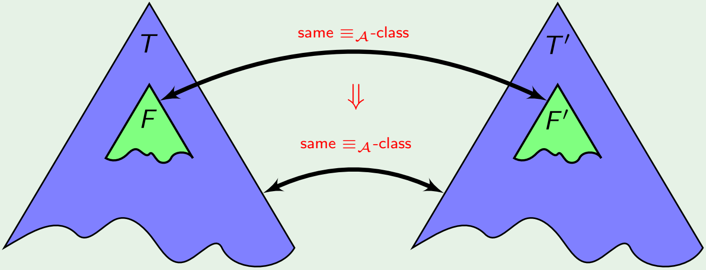
- $\equiv_{\mathcal{A}}$ has *finite index*
- if $T_1 \equiv_{\mathcal{A}} T_2$, then $\mathcal{A}$ generates *similar computations* on T_1 and T_2.

Proposition

The equivalence relation $\equiv_{\mathcal{A}}$ is compatible with tree substitutions.

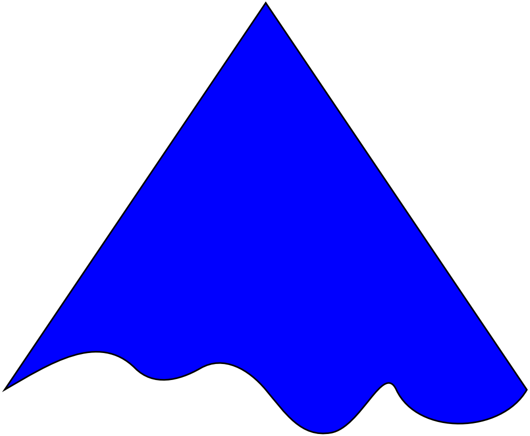
Intuitive explanation

Consider a tree T and a **factor** F inside it.
Take F' such that $F' \equiv_{\mathcal{A}} F$ and let $T' := T[\![F/F']\!]$.
Then $T' \equiv_{\mathcal{A}} T$.



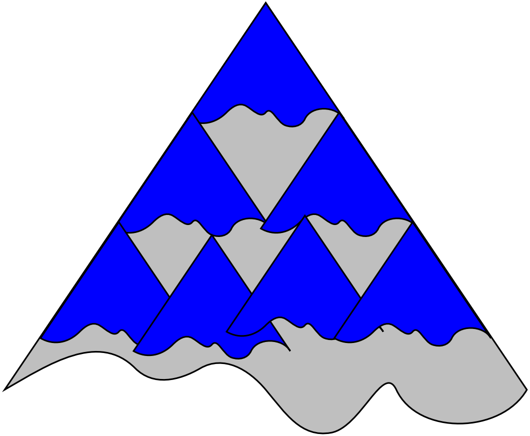
Contractions

$\Rightarrow$ *We can replace any portion of a tree* T *with its* $\equiv_{\mathcal{A}}$*-class ...*



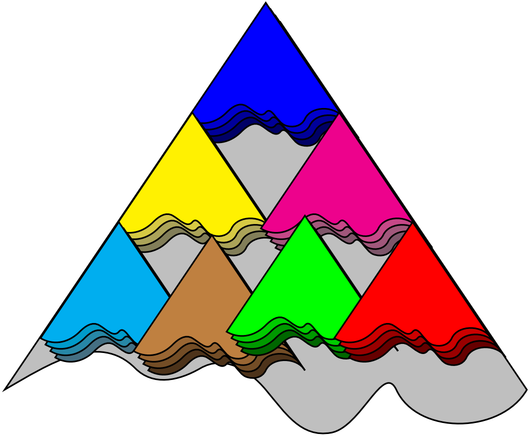
$\Rightarrow$ *We can replace any portion of a tree T with its $\equiv_{\mathcal{A}}$-class ...*

1. we decompose T into **factors**



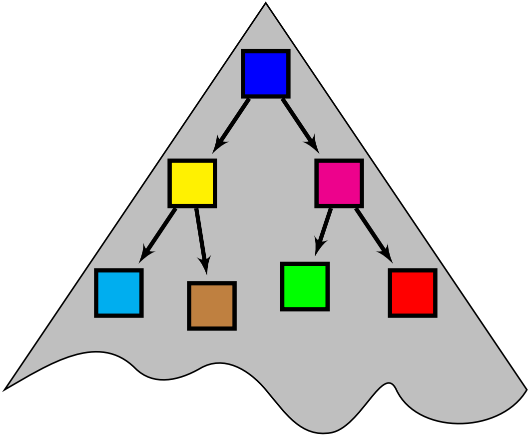
$\Rightarrow$ *We can replace any portion of a tree T with its $\equiv_{\mathcal{A}}$-class ...*

1. we decompose T into **factors**
2. we associate to each factor its **equivalence class** w.r.t. $\equiv_{\mathcal{A}}$ (these classes can be thought of as colors!)



$\Rightarrow$ *We can replace any portion of a tree T with its $\equiv_{\mathcal{A}}$-class ...*

1. we decompose T into **factors**
2. we associate to each factor its **equivalence class** w.r.t. $\equiv_{\mathcal{A}}$ (these classes can be thought of as colors!)
3. we reason on the abstracted tree $\vec{T}$, called $\mathcal{A}$-**contraction**.



Theorem (Main result)

Given an automaton $\mathcal{A}$, a tree T, and its $\mathcal{A}$-contraction $\vec{T}$ one can build an automaton $\vec{\mathcal{A}}$ such that

$$\vec{T} \in \mathscr{L}(\vec{\mathcal{A}}) \qquad \textit{iff} \qquad T \in \mathscr{L}(\mathcal{A}).$$

Theorem (Main result)

Given an automaton $\mathcal{A}$, a tree T, and its $\mathcal{A}$-contraction $\vec{T}$ one can build an automaton $\vec{\mathcal{A}}$ such that

$$\vec{T} \in \mathscr{L}(\vec{\mathcal{A}}) \qquad \textit{iff} \qquad T \in \mathscr{L}(\mathcal{A}).$$

Proof idea

Define $\vec{\mathcal{A}}$ in such a way that it *mimics* the computations of $\mathcal{A}$ on T at a "coarser level":

- the input alphabet of $\vec{\mathcal{A}}$ is the set of all $\equiv_{\mathcal{A}}$-classes
- the states of $\vec{\mathcal{A}}$ encode the finite amount of information processed by $\mathcal{A}$ up to a certain point,
- the transitions of $\vec{\mathcal{A}}$ compute new states by "merging" the information of the current state with the information provided by the input symbol (i.e., the $\equiv_{\mathcal{A}}$-class of the current factor).

Corollary

If a tree T *has a regular* $\mathcal{A}$*-contraction* $\overrightarrow{T}$*,*
then one can decide whether $T \in \mathscr{L}(\mathcal{A})$*.*

Corollary

If a tree T has a regular $\mathcal{A}$-contraction $\vec{T}$,
then one can decide whether $T \in \mathscr{L}(\mathcal{A})$.

... We can also iterate contractions on a tree T
in oder to decide whether $T \in \mathscr{L}(\mathcal{A})$!

Example

If T has an $\mathcal{A}$-contraction $\vec{T}$
and $\vec{T}$ has a *regular* $\vec{\mathcal{A}}$-contraction $\vec{\vec{T}}$

then we can decide if $\vec{\vec{T}} \in \mathscr{L}(\vec{\vec{\mathcal{A}}})$, $\vec{T} \in \mathscr{L}(\vec{\mathcal{A}})$, and $T \in \mathscr{L}(\mathcal{A})$.

Definition

It comes natural to define a

hierarchy of reducible trees:

- **rank** 0 **trees** := regular trees
- **rank** $n+1$ **trees** := trees enjoying a rank n $\mathcal{A}$-contraction, for any automaton $\mathcal{A}$.

Corollary

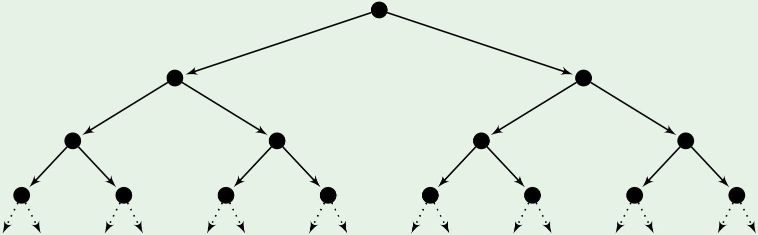
The acceptance problem (and hence the model-checking problem) of any reducible tree is decidable.

Theorem

Rank n trees are closed under **inverse forward rational mappings**.

Example

Consider the infinite $\{left, right\}$-labeled tree T_2.

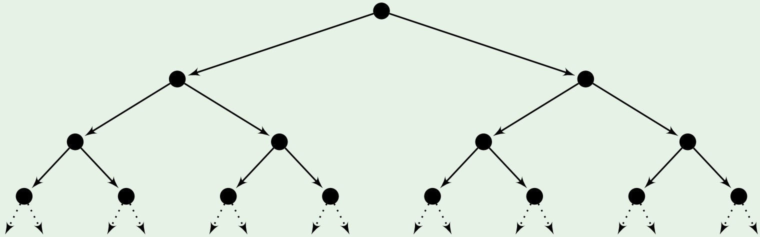


Theorem

Rank n trees are closed under **inverse forward rational mappings**.

Example

Consider the infinite $\{left, right\}$-labeled tree T_2.



We can describe the infinite $\{a, b, c\}$-labeled tree T_3 *inside* T_2:

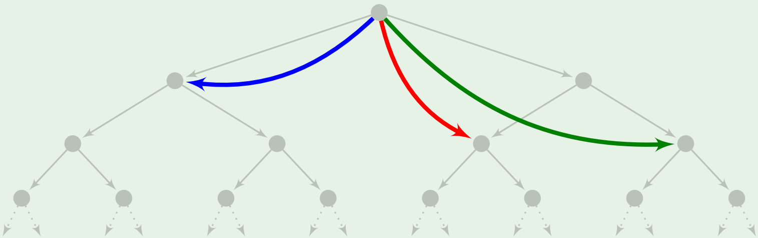
$$L_a = \{left\} \qquad L_b = \{right.left\} \qquad L_c = \{right.right\}$$

Theorem

Rank n trees are closed under
inverse forward rational mappings.

Example

Consider the infinite $\{left, right\}$-labeled tree T_2.



We can describe the infinite $\{a, b, c\}$-labeled tree T_3 *inside* T_2:

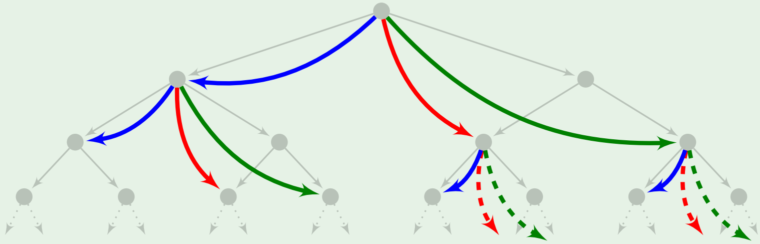
$$L_a = \{left\} \qquad L_b = \{right.left\} \qquad L_c = \{right.right\}$$

Theorem

Rank n trees are closed under **inverse forward rational mappings**.

Example

Consider the infinite $\{left, right\}$-labeled tree T_2.



We can describe the infinite $\{a, b, c\}$-labeled tree T_3 *inside* T_2:

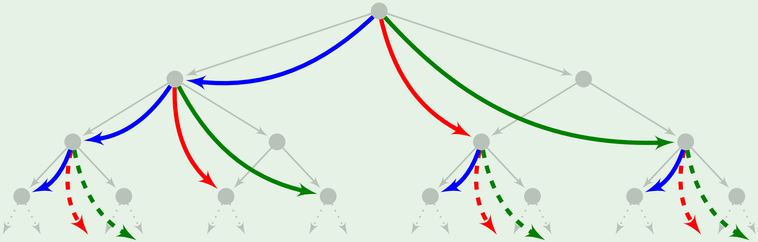
$L_a = \{left\}$ $\quad L_b = \{right.left\}$ $\quad L_c = \{right.right\}$

Theorem

Rank n trees are closed under **inverse forward rational mappings**.

Example

Consider the infinite $\{left, right\}$-labeled tree T_2.

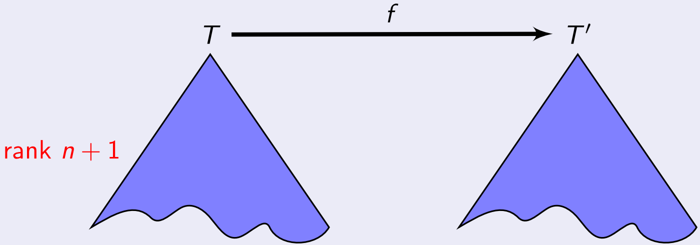


We can describe the infinite $\{a, b, c\}$-labeled tree T_3 *inside* T_2:

$L_a = \{left\}$ $\quad L_b = \{right.left\}$ $\quad L_c = \{right.right\}$

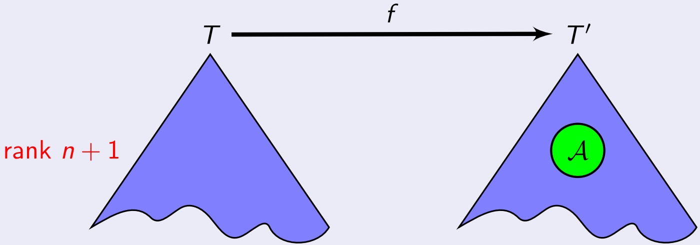
Proof idea

By induction on the rank n of the tree T (case $n = 0$ is trivial...).



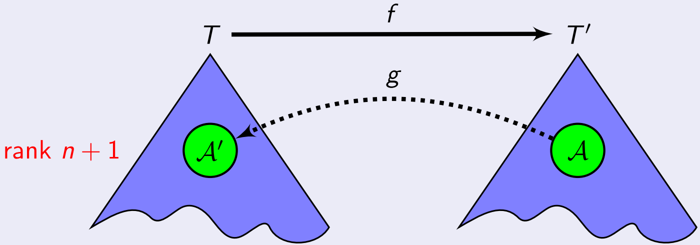
Proof idea

By induction on the rank n of the tree T (case $n = 0$ is trivial...).



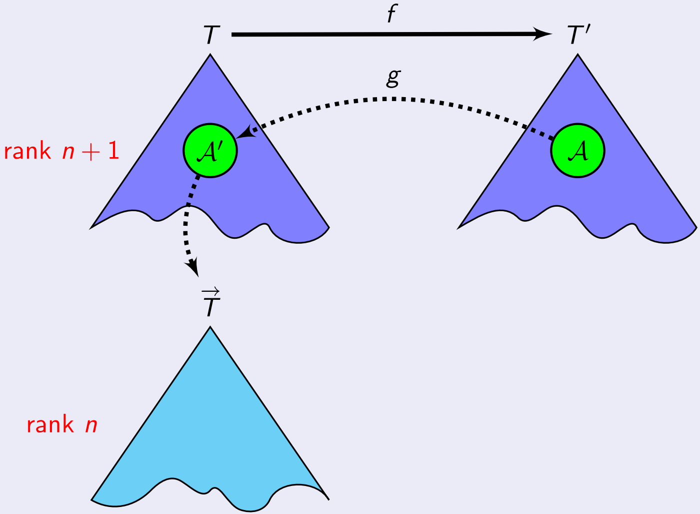
Proof idea

By induction on the rank n of the tree T (case $n = 0$ is trivial...).



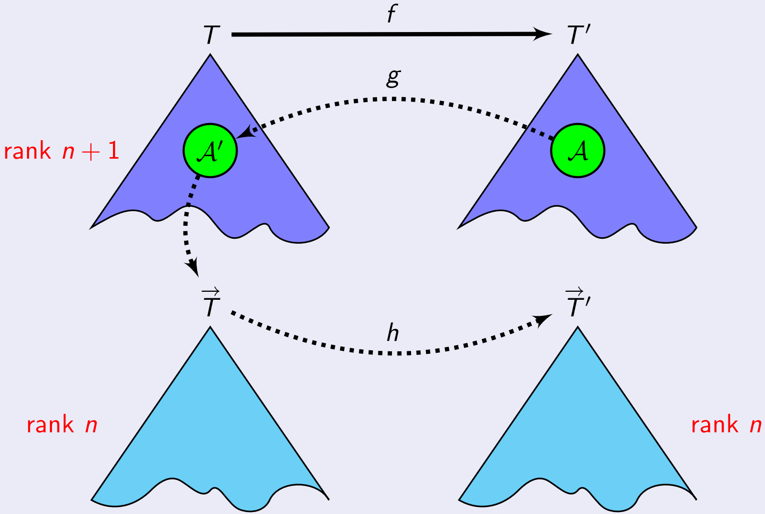
Proof idea

By induction on the rank n of the tree T (case $n = 0$ is trivial...).



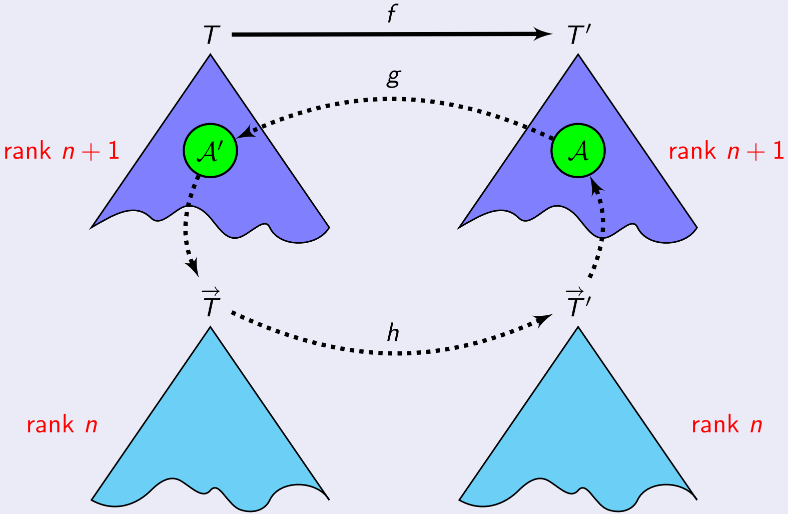
Proof idea

By induction on the rank n of the tree T (case $n = 0$ is trivial...).



Proof idea

By induction on the rank n of the tree T (case $n = 0$ is trivial...).



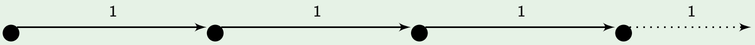
Theorem

The class of reducible trees is closed under the operation of **unfolding with backward edges and loops** $\mathcal{F}lip\mathcal{U}nfolding$.

More precisely, for every $n \in \mathbb{N}$,
if T *is a rank* n *tree, then* $\mathcal{F}lip\mathcal{U}nfolding(T)$ *is a rank* $n+1$ *tree.*

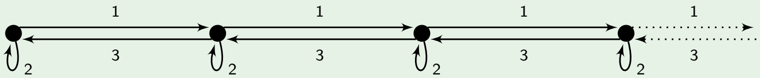
Proof by example

As a simple case, consider the **semiinfinite line** L (a *rank* 0 tree). We have to show that $T = \mathcal{F}lip\mathcal{U}nfolding(L)$ is a *rank* 1 tree.



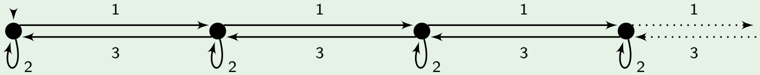
Proof by example

As a simple case, consider the **semiinfinite line** L (a *rank* 0 tree).
We have to show that $T = \mathcal{F}lip\mathcal{U}nfolding(L)$ is a *rank* 1 tree.



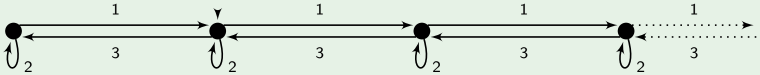
Proof by example

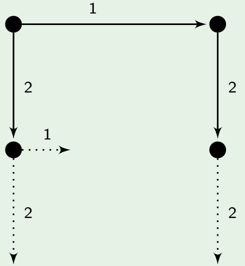
As a simple case, consider the **semiinfinite line** L (a *rank* 0 tree). We have to show that $T = \mathcal{F}lip\mathcal{U}nfolding(L)$ is a *rank* 1 tree.



Proof by example

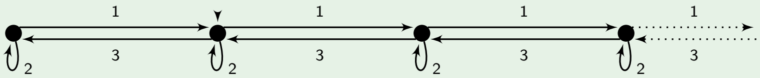
As a simple case, consider the **semiinfinite line** L (a *rank* 0 tree). We have to show that $T = \mathcal{F}lip\mathcal{U}nfolding(L)$ is a *rank* 1 tree.

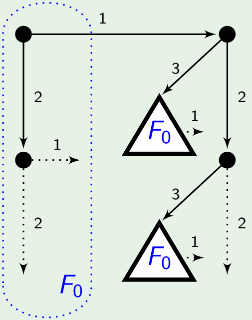




Proof by example

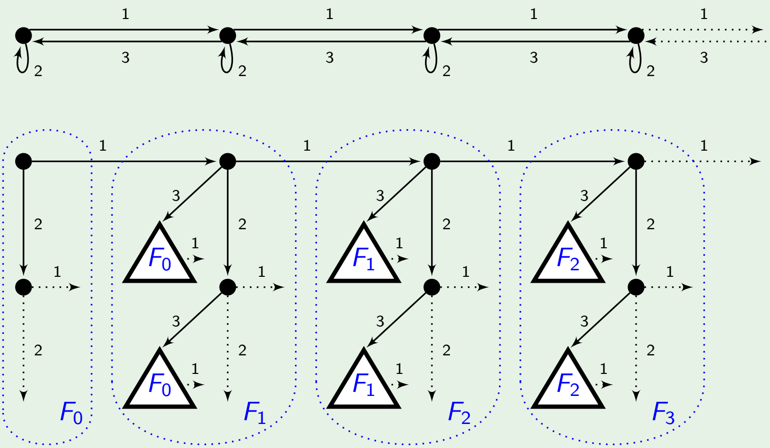
As a simple case, consider the **semiinfinite line** L (a *rank* 0 tree).
We have to show that $T = \mathcal{F}lip\mathcal{U}nfolding(L)$ is a *rank* 1 tree.





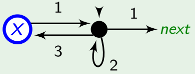
Proof by example

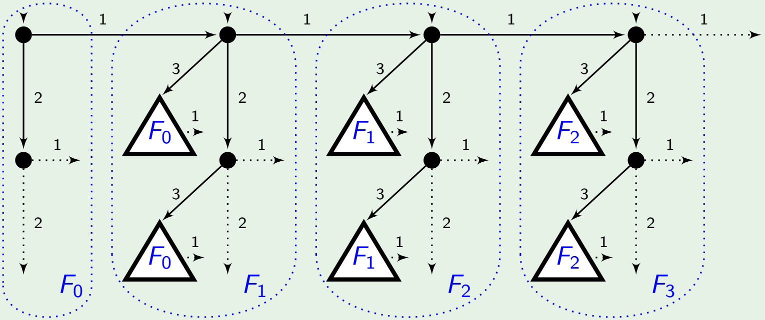
As a simple case, consider the **semiinfinite line** L (a *rank* 0 tree).
We have to show that $T = \mathcal{F}lip\mathcal{U}nfolding(L)$ is a *rank* 1 tree.



Proof by example

Every factor is obtained from its predecessor via a *substitution*:

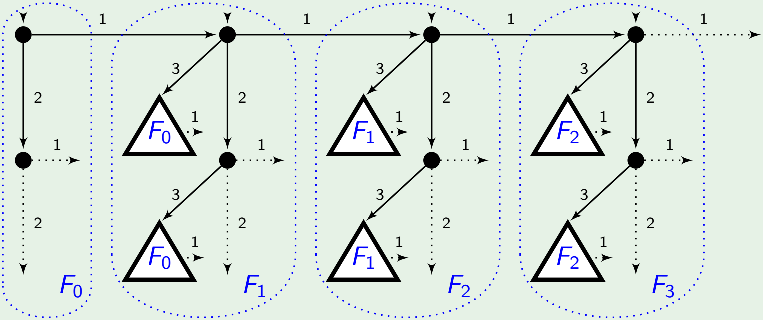
$F_{n+1} = \mathcal{U}nfolding\Big($  $\Big)[\![x/F_n]\!]$.



Proof by example

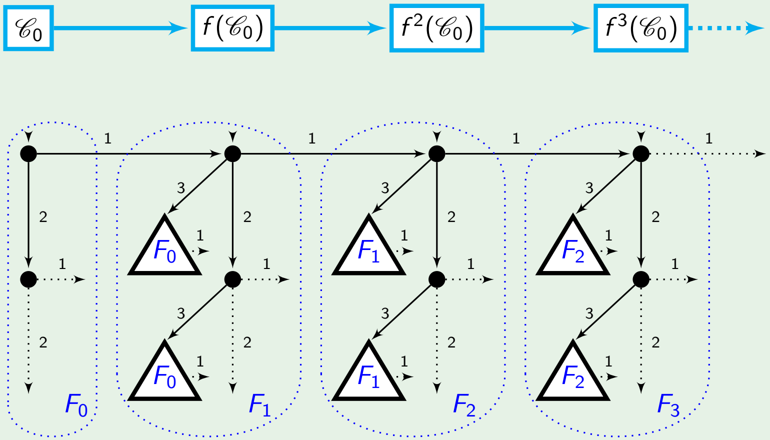
Since $\equiv_{\mathcal{A}}$ is a *congruence* with respect to substitutions, the sequence of the $\equiv_{\mathcal{A}}$-classes $\mathscr{C}_0, \mathscr{C}_1, \mathscr{C}_2, \ldots$ of factors $F_0, F_1, F_2, \ldots$ can be recursively characterized as follows:

$$\begin{cases} \mathscr{C}_0 = \left[F_0\right]_{\equiv_{\mathcal{A}}} \\ \mathscr{C}_{n+1} = f\left(\mathscr{C}_n\right) \qquad \text{(for a suitable function } f\text{)} \end{cases}$$



Proof by example

From Pigeonhole Principle,
the $\mathcal{A}$-contraction $\vec{T}$ of T is a *rank* 0 (i.e., regular) tree
$\Rightarrow$ T is a rank 1 tree.



Theorem

All deterministic trees of the **Caucal hierarchy** *can be obtained from regular trees via inverse forward rational mappings and unfoldings with backward edges and loops.*

$$\mathtt{Caucal}_0 = \{ T \ : \ T \text{ deterministic regular tree} \}$$

$$\mathtt{Caucal}_{n+1} = \left\{ f\left(\mathcal{F}lip\mathcal{U}nfolding(T)\right) \ : \ \begin{array}{l} T \in \mathtt{Caucal}_n, \\ f \text{ inverse forward mapping} \end{array} \right\}$$

Theorem

All deterministic trees of the **Caucal hierarchy** *can be obtained from regular trees via inverse forward rational mappings and unfoldings with backward edges and loops.*

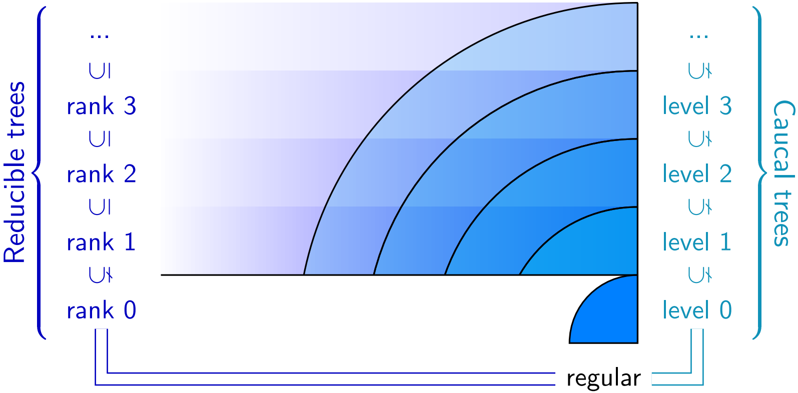
$$\mathtt{Caucal}_0 = \{\, T \;:\; T \text{ deterministic regular tree} \}$$

$$\mathtt{Caucal}_{n+1} = \left\{ f\big(\mathcal{F}lip\mathcal{U}nfolding(T)\big) \;:\; \begin{array}{l} T \in \mathtt{Caucal}_n, \\ f \text{ inverse forward mapping} \end{array} \right\}$$

Corollary

The reducible trees include all deterministic trees of the Caucal hierarchy: $\mathtt{Rank}_n \supseteq \mathtt{Caucal}_n$ *for all* $n \in \mathbb{N}$.

Actually, the inclusion is proper for each level:



Other results

- Characterization of languages recognized by **two-way alternating tree automata**
- Decidability of MSO theories of **morphic trees**

Other results

- Characterization of languages recognized by **two-way alternating tree automata**
- Decidability of MSO theories of **morphic trees**

Open problems

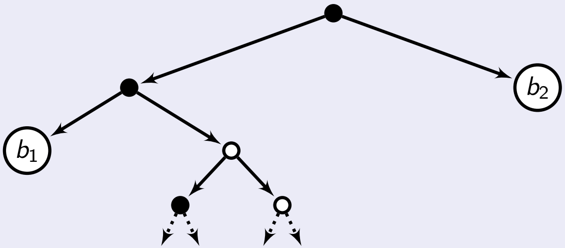
- To establish whether the hierarchy of reducible trees is *strictly increasing* or not
- To capture trees generated by **higher-order recursive program schemes**
- To generalize the approach towards *colored graphs*.

Definition

The $\equiv_{\mathcal{A}}$-class of a (marked) tree T
is represented by a *set of triples* of the form

$$\left(\begin{array}{c} R(\texttt{root}) \\ \{\mathcal{I}nf\mathcal{O}cc(R|\pi) \,:\, \pi \text{ branch of } F\} \\ \{(F(w), R(w), \mathcal{O}cc(R|w)) \,:\, w \text{ leaf of } F\} \end{array}\right)$$

over *all possible partial runs* R of $\mathcal{A}$ on F.

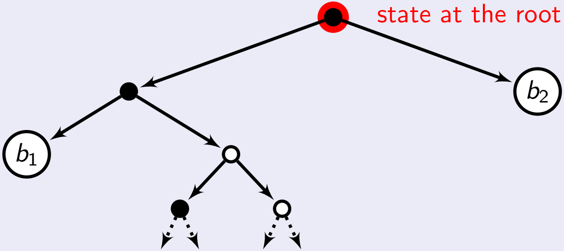


Definition

The $\equiv_{\mathcal{A}}$-class of a (marked) tree T
is represented by a *set of triples* of the form

$$\begin{pmatrix} R(\texttt{root}) \\ \{\mathcal{I}nf\mathcal{O}cc(R|\pi) \,:\, \pi \text{ branch of } F\} \\ \{(F(w), R(w), \mathcal{O}cc(R|w)) \,:\, w \text{ leaf of } F\} \end{pmatrix}$$

over *all possible partial runs* R of $\mathcal{A}$ on F.

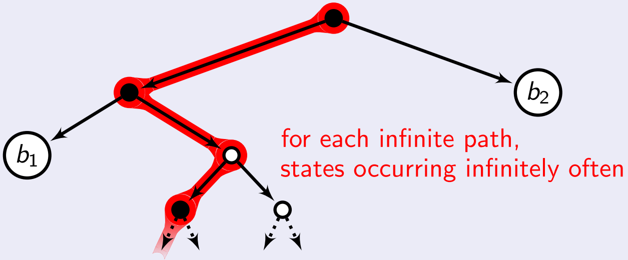


Definition

The $\equiv_{\mathcal{A}}$-class of a (marked) tree T
is represented by a *set of triples* of the form

$$\left(\begin{array}{c} R(\texttt{root}) \\ \{\mathcal{I}nf\mathcal{O}cc(R|\pi) \;:\; \pi \text{ branch of } F\} \\ \{(F(w), R(w), \mathcal{O}cc(R|w)) \;:\; w \text{ leaf of } F\} \end{array} \right)$$

over *all possible partial runs* R of $\mathcal{A}$ on F.

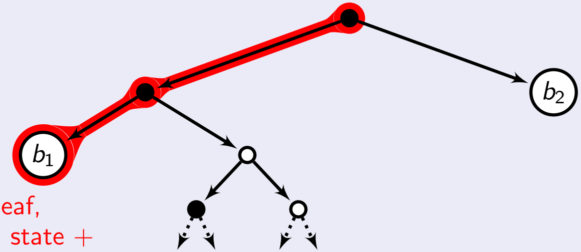


Definition

The $\equiv_{\mathcal{A}}$-class of a (marked) tree T
is represented by a *set of triples* of the form

$$\left(\begin{array}{c} R(\texttt{root}) \\ \{\mathcal{I}nf\mathcal{O}cc(R|\pi) \,:\, \pi \text{ branch of } F\} \\ \{(F(w), R(w), \mathcal{O}cc(R|w)) \,:\, w \text{ leaf of } F\} \end{array} \right)$$

over *all possible partial runs* R of $\mathcal{A}$ on F.



for each leaf,
marker + state +
states along access path